Improved Touch-screen Inputting Using Sequence-level Prediction Generation

Xin Wang, Xu Li, Jinxing Yu, Mingming Sun, Ping Li
Cognitive Computing Lab
Baidu Research
No.10 Xibeiwang East Road, Beijing, China
10900 NE 8th St. Bellevue, WA 98004, USA
{wangxin60,lixu13,yujinxing,sunmingming01,liping11}@baidu.com

ABSTRACT

Recent years have witnessed the continuing growth of people's dependence on touchscreen devices. As a result, input speed with the onscreen keyboard has become crucial to communication efficiency and user experience. In this work, we formally discuss the general problem of input expectation prediction with a touch-screen input method editor (IME). Taken input efficiency as the optimization target, we proposed a neural end-to-end candidates generation solution to handle automatic correction, reordering, insertion, deletion as well as completion. Evaluation metrics are also discussed base on real use scenarios. For a more thorough comparison, we also provide a statistical strategy for mapping touch coordinate sequences to text input candidates. The proposed model and baselines are evaluated on a real-world dataset. The experiment (conducted on the PaddlePaddle deep learning platform[1]) shows that the proposed model outperforms the baselines.

ACM Reference Format:
Xin Wang, Xu Li, Jinxing Yu, Mingming Sun, Ping Li. 2020. Improved Touch-screen Inputting Using Sequence-level Prediction Generation. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3366423.3380080

1 INTRODUCTION

In most mobile devices, the soft-keys are small and lack tactile feedback of physical key boundaries. It is hence fairly easy for touches to fall out of the visible soft-button areas of expected characters. Once, an unexpected character sequence has been typed, it is difficult to move the cursor to the precise location. Thus, one has to delete most of the input sequence from the end and re-input it, which decreases input efficiency and leads to bad user experiences.

To minimize the need for manual correction, these days touch-screen input methods generate several possible candidates for the input sequence from the touch coordinates in real-time. As shown in Figure 1, when the user touches the soft keyboard, the application continuously update a list of the candidate words, until the user selects one word from the list. This has become the common practice in mobile devices. There are, however, at least three major challenges in such a candidate generation and display process.

[1]https://www.paddlepaddle.org.cn

ACM ISBN 978-1-4503-7023-3/20/04.
https://doi.org/10.1145/3366423.3380080

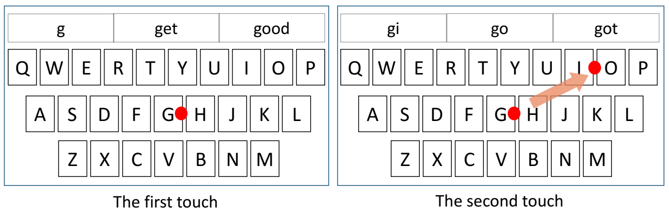

Figure 1: An illustration of the input process and the corresponding candidate lists. The first touch falls into the area of 'g', so the candidate list shows high-probability words beginning with 'g'. Then the second touch falls on the board of 'i' which is close to 'o', so the candidate list contains both possible strings 'gi' and 'go'.

First of all, there are various **mistaken inputs** in the typing process. (i): The touch coordinates may fall outside the soft-button areas of target characters [26], and the distributions vary according to the different characters. Besides, for the same character, different context also leads to inconsistent coordinate distributions. (ii): Similar pronunciation also causes the touch distribution diffusion [8]. For instance, people misuse 'c' and 'k' because they may have the same syllable. (iii): Moreover, when typing with two hands, people sometimes input the sequence in the wrong order. For example, one can input 'doing' as 'doign' for the over-rapid typing of the left hand. Besides, (iv): missing typing or (v): redundant typing operations also make it challenging to cover expected input with candidates whose character numbers are equal with the touch coordinate numbers.

Secondly, a coordinate sequence may be an **incomplete input** concerning the expectation. For example, when the input method receives the coordinate sequence of 'sat', the user's expected word may be 'Saturday' or 'satisfied'. Ideally, an algorithm should automatically complete the input and update the candidate list to help users finish typing with one selection. Therefore, it should be obvious that the candidates with only character-level corrections are far from being enough to satisfy user expectations.

Last but not least, the sizes of mobile device screens are small, and hence only **limited candidates** can be displayed on the list. The generated candidates should be scored and ranked properly.

To address the above challenges, in this paper we introduce the neural encoder-decoder framework with a beam search. It takes noisy coordinates as input to generate scored candidates. The neural sequence-to-sequence models have the promising potential to handle symbol mapping, inserting, deleting and reordering, which has been proved in translation tasks [2]. These properties are helpful to correct the **mistaken input**. Moreover, the encoder-decoder framework has achieved remarkable performance on image captioning or automatic reply [24, 31]. It shows that such a framework has the capacity to learn sequence-level representation tasks that do not rely on one-to-one symbol mapping. Thus, it is a good choice to encode the coordinate sequence of the first few characters (relatively **incomplete input**) and generate the completed sequence. During a beam search decoding process, we obtain the scores of possible sequences according to the softmax probabilities, which can be used to select **limited candidates** to be displayed. In this way, the input efficiency of users can be improved by avoiding manual correction and completion.

The contributions of this work can be summarized as follows:

- We formally describe the task of candidate list prediction on touch screen devices and the optimization objective for input efficiency improving. The proposed framework is trained incrementally and validated with different measures on a real-world dataset. The experimental results show that the proposed methods significantly improve the top-1 accuracy and the character-per-touch metric over baseline strategies.
- For a simplified prediction problem, we introduce a neural sequence-to-sequence model that takes the touch coordinate sequence as input and generates words by handling replacing, reordering, inserting, deleting and completing in an end-to-end manner.
- Two evaluation metrics are explained based on real-world application scenarios. We also discuss the inconsistency of the two metrics and the underlying difference of assumptions of user preferences.
- For a more thorough comparison, we provide a statistical strategy for mapping touch coordinate sequences to word-level input candidates as a baseline.

2 PROBLEM STATEMENT

We examine the user behavior of text input on mobile devices through a commercial input method application **Facemoji**. The task of continuously updating the candidate list can be formally described as follow. The core function of the algorithm can be viewed as an agent who predicts a list of candidate words $W_{i,t} = \left\langle w_{i,t}^1, w_{i,t}^2, ..., w_{i,t}^n \right\rangle$ after the user inputs the t-th character of the i-th word based on the user inputs (P_{i-1} and C_i), what the user can see in the interface (L and $\mathcal{W}_i$) and the user's habits (U).

$$W_{i,t} = f_\theta(P_{i-1}, C_i, L, \mathcal{W}_i, U) \tag{1}$$

where $P_{i-1} = (p_1, p_2, ..., p_{i-1})$ is the previous words inputted by the user, $C_i = (c_i^1, c_i^2, ..., c_i^t)$ is the inputted touch coordinates of the current word, L represents the keyboard layout that maps the coordinates to characters, $\mathcal{W}_i = \left\{W_{i,j} | 1 \leqslant j \leqslant t-1\right\}$ indicates the candidate lists of words that have been shown to the user right after she/he inputs the former j-th coordinates, and U represents user specific features.

There are multiple objectives for developing input method algorithms. For example, some applications focus on entertaining aspects and put emojis on the top of the list. In this work, we focus on the input efficiency. If a character sequence in the candidate list is selected by the user, we call such sequence a user expectation X_i. Our objective is to optimize parameters θ for function f_θ to minimize the position of X_i in list $W_{i,t}$.

$$\arg\min_\theta pos(X_i, W_{i,t}) \tag{2}$$

where $pos(A, B)$ represents A's particular rank position in list B.

2.1 Practical Concerns for the Problem

Due to increasing privacy concerns, we only randomly sample limited information during data collection. Firstly, we only extract the touch coordinates within sampled words and collect no inter-word information. With absence of P_{i-1}, we cannot make use of a language model as previous works [13, 32]. Secondly, the stream data is anonymized and randomly shuffled, and no temporal, geographical information or password is included. Thus, no personalized information U is used although it has been proved to be helpful [13, 30]. Also, limited by the size of the log stream, we cannot save candidate lists for all time steps $\mathcal{W}_i$. We hence only collect the coordinate sequence C_i and user expectation of X_i. The word prediction problem is thus simplified as:

$$W_{i,t} = f_\theta(C_i, L) \tag{3}$$

Note that the result is independent with context word. From now on, we will omit the subscript i in the denotations.

2.2 Online Solution and Data Collection

Given the coordinate sequence C, there are many methods to predict the candidate word list. The online solution is a rule-based one designed by experts. It gets the original user input based on which rectangle areas of characters the coordinates fall into. And then the solution generates candidates by enumerating the possible character replacing, deleting, inserting and reordering and completing. Finally, the candidates are sorted based on the coordinate deviation level (including Bayesian touch distance [5]) and word frequency.

Users update the candidate list by typing, deleting and re-typing until they get their expectations. During this process, we can collect the original input coordinates C and corresponding expectation X. And the rule-based solution can be iteratively updated based on the collected data sampled from all English keyboard users.

3 FRAMEWORK FOR CONVERTING COORDINATES TO CANDIDATES

It is expensive to design different refined hand-crafted rules or develop different rules for countries with different user behaviors. Data-driven methods are promising to outperform rule-based ones and can be easily trained on datasets of different distributions (e.g., datasets from non-English speaking countries).

A recurrent neural network (RNN) is ideal for handling sequence input of variable length. Meanwhile, an RNN decoder with a beam search can naturally generate several character sequences as candidates. We train a sequence-to-sequence framework with attention using C and X as input and target sequence respectively. In this way, the model learns to generate X as the top one candidate and minimize $pos(X, W_t)$. If $pos(X, W_t) > 1$, the loss will not be zero and the parameters will continue to be updated.

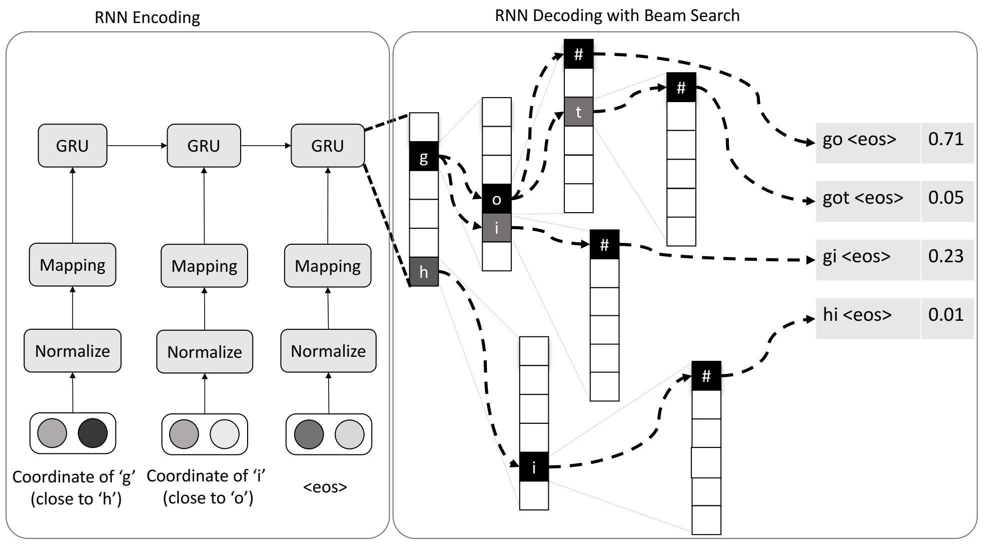

Figure 2: The illustration of the proposed framework handling coordinate input of 'gi'. Some secondary components, such as softmax layers are omitted. Both '#' and 'eos' represent end of the sequence.

3.1 Encoder and Decoder Network

Deep neural networks often take high-dimensional dense vectors as input [3]. Here we use a full connection layer to map the two-dimensional points to high-dimensional space.

$$e_t = f(Wc^t + b) \tag{4}$$

where f represents the sigmoid function. W and b represent the weight and bias respectively. An RNN structure makes it possible to encode the information of context touches step by step. We leverage the gated recurrent unit (GRU) to encode the arbitrary-length touch coordinates [10]. The hidden state vector of the encoder at time t can be computed as:

$$h_t = \mathrm{GRU}(h_{t-1}, e_t) \tag{5}$$

The length of predicted words is not always equal with the number of input coordinates. Therefore, besides automatic correction, the generator is supposed to have the capacity of deleting redundant inputs, inserting missing inputs and completing the input sequences. The RNN-decoder can generate a flexible-length sequence to cover the above situations. Thus, RNN decoder with attention mechanism [2, 27] is introduced to learn weights for the crucial inputting and handle possible alignment:

$$d_t = GRU(d_{t-1}, v_{t-1}, w_t) \tag{6}$$

where v_{t-1} is the embedding of the character predicted in last time step, and attention vector w_t can be computed as:

$$u_{tj} = v^T \sigma_h(W \cdot [h_t, d_j]) \tag{7}$$

$$a_{tj} = softmax(u_{tj}) \tag{8}$$

$$w_t = \sum_{j=1}^{T_x} a_{tj} h_j \tag{9}$$

where σ_h represents ReLU function [14], while W and v^T are learnable parameters. The output characters are select based on the softmax probabilities [7].

$$y_t = softmax(d_t) \tag{10}$$

And the cross-entropy loss is used to measure the difference between the predicted distributions and the gold standard ones [11].

3.2 Learning and Predicting Details

During the training process, the decoder greedily selects the character with the largest probability. Teacher-forcing technology is used to improve the training process [15]. The current RNN input is decided based on the ground-truth character of last-time, instead of the predicted one. During prediction, a beam search strategy [16] is used to i): get more accurate prediction than greedy search by enlarging search breadth and ii): generate multiple scored candidates.

4 EVALUATION METRICS

In this section, we discuss two perspectives on evaluating an inputting algorithm and the possible inconsistency.

4.1 Ranking Metric and Accuracy

Selecting the candidates to be displayed on the user interface can be treated as a ranking problem, which can be evaluated with discounted cumulative gain (DCG) [17, 22]. In the real-world scene, when users finish typing a word, they are used to touch the spacebar and expect the top one candidate with a space shown on the text-field. Therefore, DCG_1 is used to address the importance of the top one prediction. And the measurement can be written as:

$$DCG_1 = 2^{rel_1} - 1 \tag{11}$$

where the graded relevance rel_1 of all samples is set to 1 if and only if the top one candidate covers the expectation.

$$rel_1 = \begin{cases} 1, & \text{if top candidate is expected} \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

In this condition, the average top one ranking metric is the same with accuracy or word prediction rate [32]. We see that the existing word-level matching evaluation is a specialization of list-level ranking metrics.

4.2 Character per Touch

Evaluating in a character-level, the input efficiency can be represented as the ratio of the numbers of expected character to the times of the user touching screen. The previous word uses Keystroke Savings (KS) to evaluate the character-level performance in automatic completing task [13, 32]. Such a metric can be regarded as a micro-averaged evaluation on all character in the test set. However, in practice, we try to find every extreme bad case that hurts the user experience. Therefore, we evaluate the character-level performance word by word and then compute macro-averaged result upon all words.

Moreover, simulations in previous works assume that (a) the user will notice immediately if the target word is among top candidate list and (b) the characters have been inputted is exactly same with user expectation. However, the collected log shows that i) users select the expected word after some redundant touches and ii) all existing candidates may contain character different from user expectation (but the same with user's mistouch).

Instead of using character-level simulation, we perform an evaluation based on the word-log data. Formally, given the user expectation X and coordinate sequence C_t, if the algorithm returns a K-length candidate list W_t, then the character per touch of candidate $w_t^k \in W_t$ can be computed as:

$$\mathrm{CPT}_{cand}(C_t, X, w_t^k) = \frac{len(X)}{t + update(X, w_t^k)} \tag{13}$$

where $update(X, w_t^k)$ denotes the number of operations that user need to delete the wrongly typed character in w_t^k and retype the correct ones in X. So the denominator is the times of user touching screen to get the expected sequence and numerator indicates the length of the user expectation.

We assume that among all candidates $w_t^k \in W_t$, the user will choose the one with the largest CPT_{cand} to avoid deleting or retyping operations. So character per touch of candidate list CPT_{list} can be computed as:

$$\mathrm{CPT}(C_t, X, W_t) = max(\mathrm{CPT}_{cand}(C_t, X, w_t^k)) \tag{14}$$

4.3 Inconsistency between the Metrics

In practice, the above ranking metric and character-touch ratio are not always consistent. CPT_{list} relies not only on whether the candidate is correct but also on the number of characters that need to be modified if it is wrong. Therefore some candidates may increase the mathematical expectation of accuracy but decrease the mathematical expectation of CPT_{list} at the same time.

If a user prefers one whole word-level correction rather than several tiny character-level corrections in different words, the sequence-level accuracy works better. Conversely, CPT_{list} measurement can bring a better user experience. Therefore, in evaluation for real-world application, only those algorithms work better on both metrics can be considered as sufficient improvements.

5 EXPERIMENT

5.1 Experimental Settings

We conduct incremental training because it is friendly to data streams of online applications. The training process stops when the number of samples reaches 50 million, and the model with the minimum Log perplexity on the streaming validation data is selected for testing. The validation set and testing set both contain 10 million random samples. We believe the training and testing set provides more than needed range and resolution of the input and output variable values, and thus cross-validation is skipped.

We set the hidden layer size to 256 for both bidirectional GRU encoder and decoder. The input coordinates are mapped to a 512-dimensional space through the full-connection layer. In the decoding process, each character is mapped to an embedding of 256 dimensions.

We set the batch size to 128 for the training, validating and testing process. The gradient is clipped to a number no larger than 5.0 and the Learning rate is fixed to 0.0001 during Adam optimization [18]. The above hyper-parameters are selected with a grid search on an appropriate scale or log scale on the validation set. Other hyper-parameters combinations in the frequently-used scales could bring at most about 3% accuracy reduction. It's an obvious impact, but such a non-optimal accuracy is still higher than baselines.

During the evaluation, the beam size is set to 3, which is enough to cover all mainstream settings of the exclusive choice list on the touchscreen mobile devices.

5.2 Baseline Methods and Comparison

To show the effectiveness of the proposed framework, we compare it with several baseline systems. The basic **Rectangle** baseline contains only a simple detection rule. Therefore, besides **Online** solution described in Section 2, we proposed a **Statistical Model** that implements correcting, reordering and completing as an enhanced baseline for a more thorough comparison.

5.2.1 Rectangle. Given a touch coordinate, we simply output the expected character according to whose rectangle area it falls in. And all the characters make up the candidate.

5.2.2 Statistical Model. Given the result of rectangle-based decision $G = \langle g_{1:t} \rangle$, we can generate possible candidates through a series of operations. Firstly, we replace rectangle-decided character g_k with possible expected character m_k ($1 \leqslant k \leqslant t$) and get refined candidates $M = \langle m_{1:t} \rangle$. Then for each adjacent character pair m_k and m_{k+1} ($1 \leqslant k \leqslant t-1$), we get both original and reorder sequences, denoted as R. For each R, we get gain candidates by lookup the high-frequency words that contain the prefix R. The final candidate set $\mathcal{F}(R)$ can be represented as:

$$\mathcal{F}(R) = \{F | prefix(F) = R\} \tag{15}$$

The probability of final candidate $F \in \mathcal{F}(R)$ can be estimated according to the above generation processes:

$$p(F) = p(F|R) \sum_{M \in \mathcal{M}} p(R|M) p(M|G) \tag{16}$$

$p(F|R)$ can be computed with maximum likelihood estimation on a dictionary with word frequency.

We assume that the mistouch and same-pronunciation misuse of characters are independent with context inputs. Given the original sequence G, the probability of a modified sequence M can be computed with the product of the probability of character-to-character replacing.

$$p(M|G) = \prod_{n=1}^{N} p(m_n|g_n) \tag{17}$$

Similarly, with the independence assumption, the reorder probability can be described as:

$$p(R|M) = \prod_{n=1}^{N-1} p(r_n r_{n+1}|m_n m_{n+1}) \tag{18}$$

The above model can generate candidates as well as get its probabilities. However, a more general score can be estimated by the log-linear model without assuming that the probabilities are equally important [25]. In this work, we train a log-linear model to score the generated candidates by using replacing (modifying), reordering and completing probabilities as features. Such a strategy implements automatic correction as well as automatic completion, which forms a strong baseline for the neural model.

Table 1: Accuracy of different methods.

	Accuracy (DCG_1)	CPT_{list}
Rectangle	0.693	0.938
Online	0.803	1.020
Statistical Model	0.814	1.022
Neural Model	**0.859**	**1.026**

5.2.3 Experimental Results. The experimental results of the ranking metric and character-per-touch metric are shown in Table 1. The online solution, statistical model and neural model outperform the Rectangle baseline. It is because these three methods alleviate the "fat finger problem" [26] by automatic correction. Online, statistical and neural approaches can reach a CPT_{list} larger than 1. It indicates that the automatic completions improve input efficiency. Nevertheless, the improvement of performance of the online solution is limited. We will see that data-driven models make better use of comprehensive information in data than expert-designed rules.

The neural model outperforms the statistical model on both accuracy and CPT_{list} measures. The statistical approach can generate candidates with out-of-rectangle characters by enumerating the character-replacing operation and achieve good coverage of the expected word. However, the model cannot represent the real probabilities because of a lack of coordinate-to-character (mapping) probabilities. These probabilities are memory consuming for mobile devices. Therefore, with only character-level features, the statistical strategy is mainly dominated by word frequency. In contrast, the neural model generates candidates by considering various factors. Both word frequency and coordinate-based probability are encoded in the network parameter during training.

The difference of CPT_{list} among the online solution, statistical model and the neural model is not as evident as the difference of accuracy. In the online solution or statistical model, although the expected inputs do not match the top one candidate of the statistical model, they are probably covered by other top candidates. Therefore, the maximal value of multiple CPT_{cand} results of these models is the same at most time. The Rectangle method fails to generate more candidates out of the touching areas and needs more touches to delete, re-input and complete the expected character, which results in a lower CPT_{list} value less than 1.

According to the statistics result, the major operations needed in the data are correction, completion, and insertion (of apostrophe) whose proportions are 14.18%, 8.86% and 4.04% respectively. In comparison, there are only 0.28% samples need reordering and 0.08% samples need deleting. We can see that having the capacity to insert a character into original input is also a major advantage for a neural model comparing to the statistical model.

6 CASE STUDY

Table 2 demonstrates cases where auto-correction predictions are different from that of Rectangle decision. The Rectangle baseline fails in both making use of character-level context information and completing the character sequences.

Table 2: Examples of top one candidate of different methods.

	Rectangle	Online	Statistical	Neural	Expected
1	toda	today	today	today	today
2	perosn	person	person	person	person
3	fone	gone	fine	fine	fine
4	live	love	love	live	live
5	bit	but	but	bit	bit

Both the rule-based online solution and the statistical method can generate candidates covering different situations including completing (#1), reordering (#2) and replacing (#3-#5). However, we find that the results of these two models are dominated by word frequency. While the neural model can avoid some frequent words that have a character far from the coordinate.

As discussed in Section 3, the neural framework has the capacity of generating candidates whose lengths may be unequal to the number of input touches. The input coordinate sequence is encoded in the bidirectional GRU, and the decoder generates the candidate characters without necessarily aligned to a particular touch.

Table 3: Examples of cases where the length of the predicted sequence is not equal to the number of input touches.

Operation	Rectangle	Neural
Insertion	theyre	they're
	dont	don't
Deletion	readyy	ready
	feell	feel
Completion	youn	young
	stopp	stopped

Table 3 shows some cases where the proposed framework successfully predicts the expected words whose lengths are different with numbers of input touches. The learned patterns can be summarised as following functions. i): Insertion: The sequence-to-sequence model can effectively learn to insert an apostrophe before abbreviation. However, without the sentence-level context, it is still difficult to decide whether an apostrophe should be added to the sequence of 'were'. ii): Deletion: The EOS symbols are predicted when the already predicted characters compose a frequent word. The redundant touch coordinate rarely influences the sequence representation. iii): Completion: Un-inputted characters can be predicted uninterruptedly until the decoding result form a frequent word. The network has learned to map the first few touches to several possible longer candidates.

7 COMPATIBILITY ON MOBILE DEVICES

In real-world input method application, the memory cost of an algorithm should also be taken into consideration. According to the space occupied in the device, the distributable version of the trained neural model requires only 8 megabytes of storage space and hence easy to be transmitted. When loaded by the mobile devices, 8-megabyte memory cost is fairly small compared to the gigabyte-level random access memory (RAM) of nowadays mobile devices. It is more memory-efficient than dictionary-based strategies, such as the statistical model. The prefix-to-word dictionary in direct translation strategy costs at least 25 megabytes of memory to achieve adequate candidate coverage, let alone the additional reverse dictionary if using the noisy-channel model.

The computational sources needed in the statistical model and neural model are similar. The prefix-candidates size in the statistical model and square of hidden size in the neural model are both at the level of 10 thousand. Therefore, both methods achieve reasonable average latency (less than 20ms per touch).

Finally, the framework can be deployed in devices with mobile versions of deep learning frameworks. And the installation or updating package (including trained models) is shippable through the cellular network.

8 RELATED WORKS

8.1 Input Efficiency Improving

Some recent works made use of context (by introducing n-gram or other language models) or personalized information [13, 30, 32]. These features are proven to be helpful to improve user experience. In this work, we focus on the situation with no word-level context and personal information available. In this way, this paper has discussed a framework that can be used with growing privacy concerns. Hand postures and speed or pressure along a touch-trail can also be important features [30]. The usage of such features is a valuable direction of further improving the input efficiency.

Some works try to improve the user experience by generating candidates with expert-edited correction operation and hand-craft features [19]. Instead of enumerating the correction operation and setting thresholds, the proposed models learn to score the candidates with data-driven models. There are also works choose characters with statistical criterion. The online system has made use of Bayesian touch distance [5], while Zhang et al. [33] generate candidates with a beam search on predicted character probabilities and choose words with learning to rank models. In this work, we take a different approach by adopting a unified neural end-to-end framework, to handle insertion, deletion, completion, etc. It is also obvious that one might be able to combine this work with [33] in a real production system.

In addition, there are also researches focusing on the general automatic correction problem including but not limited to the touch-screen devices [12, 23]. Our work is different in that we implement automatic completing in both the statistical model and the neural model, which is rarely addressed in existing correction works. Besides, we prove coordinate features bring more information than characters and can be used to improve correction performance on mobile devices. We should also mention that improving interactive mode is another main research route [4, 6, 29]. Our proposed framework is orthogonal to the new interactive modes and can be used to further improve user experience.

8.2 Neural Networks

Wang et al. [28] address the principle of making minimal assumptions by end-to-end learn with a deep neural network. While recurrent based encoder-decoder with attention has been validated on different tasks in various fields [2, 9] including translation, image captioning and automatic reply generation [2, 24, 31]. This work maps the two-dimensional coordinates to high-dimensional space and introduces such a framework to predict final candidate words based on one original input of touch coordinates.

Alsharif et al. [1] improve gesture typing with LSTM, which brings inspiration to our work. Gesture typing is popular in many countries/regions. On the other hand, Languages whose characters are not on the keyboards (e.g., Chinese) are unfriendly to swiping. Thus, we explore to improve typing experience for many related users accustomed to typing whatever they input.

9 CONCLUDING REMARKS

In this paper, we have formally described the general task of mapping touch-screen coordinates to the candidate list. Setting the input efficiency as the optimization objective, we have introduced the neural sequence-to-sequence model with a beam search for this task. The proposed framework takes the coordinate sequence as input and generates scored candidates by handling replacing, reordering, inserting, deleting and completing. We have presented two metrics, corresponding specialization as well as the inconsistency of the metrics according to the realistic scenes. To show the effectiveness of the proposed framework, we provide a strong baseline by implementing a statistical model with automatic correction and completion. The proposed framework is evaluated with the streaming input touches of real-world data on both metrics. The experimental result shows improvement over the baseline approaches and effectiveness of the proposed model increasing input efficiency. In future work, we will address topics like combination cases, personalizing and swiping-based input for further improvement.

Baidu's IME products have very high DAU's (daily active users). The research of this paper was inspired while we collaborated with Baidu's product teams including Chinese IME product and Facemoji multilingual IME product. Our prior work [33] has been successfully deployed in the products. We are actively working on deploying various research works including this paper and our new research on learning to rank based on "abc-boost" tree algorithms [20, 21].

REFERENCES

[1] Ouais Alsharif, Tom Ouyang, Françoise Beaufays, Shumin Zhai, Thomas Breuel, and Johan Schalkwyk. 2015. Long short term memory neural network for keyboard gesture decoding. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2076–2080.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*. San Diego, CA.

[3] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A Neural Probabilistic Language Model. *J. Mach. Learn. Res.* 3 (2003), 1137–1155.

[4] Xiaojun Bi, Tom Ouyang, and Shumin Zhai. 2014. Both complete and correct?: multi-objective optimization of touchscreen keyboard. In *CHI Conference on Human Factors in Computing Systems (CHI)*. Toronto, Canada, 2297–2306.

[5] Xiaojun Bi and Shumin Zhai. 2013. Bayesian touch: a statistical criterion of target selection with finger touch. In *The 26th Annual ACM Symposium on User Interface Software and Technology, (UIST)*. St. Andrews, UK, 51–60.

[6] Xiaojun Bi and Shumin Zhai. 2013. Touchscreen text input. US Patent 8,405,630.

[7] Christopher M Bishop. 2006. *Pattern recognition and machine learning*. springer.

[8] Eric Brill and Robert C. Moore. 2000. An Improved Error Model for Noisy Channel Spelling Correction. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL)*. Hong Kong, China.

[9] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 1724–1734.

[10] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv:1412.3555

[11] Pieter-Tjerk de Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein. 2005. A Tutorial on the Cross-Entropy Method. *Annals OR* 134, 1 (2005), 19–67.

[12] Noura Farra, Nadi Tomeh, Alla Rozovskaya, and Nizar Habash. 2014. Generalized Character-Level Spelling Error Correction. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL)*. Baltimore, MD, 161–167.

[13] Andrew Fowler, Kurt Partridge, Ciprian Chelba, Xiaojun Bi, Tom Ouyang, and Shumin Zhai. 2015. Effects of Language Modeling and its Personalization on Touchscreen Typing Performance. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI)*. Seoul, Korea, 649–658.

[14] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep Sparse Rectifier Neural Networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*. Fort Lauderdale, FL, 315–323.

[15] Anirudh Goyal, Alex Lamb, Ying Zhang, Saizheng Zhang, Aaron C. Courville, and Yoshua Bengio. 2016. Professor Forcing: A New Algorithm for Training Recurrent Networks. In *Advances in Neural Information Processing Systems (NIPS)*. Barcelona, Spain, 4601–4609.

[16] Alex Graves. 2012. *Sequence transduction with recurrent neural networks*. Technical Report. arXiv:1211.3711

[17] Kalervo Järvelin and Jaana Kekäläinen. 2000. IR evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. Athens, Greece, 41–48.

[18] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*. San Diego, CA.

[19] Kenneth Kocienda, Greg Christie, Bas Ording, Scott Forstall, Richard Williamson, and Jerome René Bellegarda. 2012. Method, device, and graphical user interface providing word recommendations for text input. US Patent 8,232,973.

[20] Ping Li. 2009. ABC-Boost: Adaptive Base Class Boost for Multi-Class Classification. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*. Montreal, Canada, 625–632.

[21] Ping Li. 2010. Robust LogitBoost and Adaptive Base Class (ABC) LogitBoost. In *Proceedings of the Twenty-Sixth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI)*. Catalina Island, CA, 302–311.

[22] Ping Li, Christopher J. C. Burges, and Qiang Wu. 2007. McRank: Learning to Rank Using Multiple Classification and Gradient Boosting. In *Advances in Neural Information Processing Systems (NIPS)*. Vancouver, Canada, 897–904.

[23] Christian Liensberger. 2015. Context sensitive auto-correction. US Patent 9,218,333.

[24] Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, and Alan L. Yuille. 2015. Deep Captioning with Multimodal Recurrent Neural Networks (m-RNN). In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*. San Diego, CA.

[25] Franz Josef Och and Hermann Ney. 2003. A Systematic Comparison of Various Statistical Alignment Models. *Computational Linguistics* 29, 1 (2003), 19–51.

[26] Katie A. Siek, Yvonne Rogers, and Kay H. Connelly. 2005. Fat Finger Worries: How Older and Younger Users Physically Interact with PDAs. In *Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction (INTERACT)*. 267–280.

[27] Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. 2015. Grammar as a Foreign Language. In *Advances in Neural Information Processing Systems (NIPS)*. Montreal, Canada, 2773–2781.

[28] Tao Wang, David J. Wu, Adam Coates, and Andrew Y. Ng. 2012. End-to-end text recognition with convolutional neural networks. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR)*. Tsukuba, Japan, 3304–3308.

[29] Xiang Xiao, Teng Han, and Jingtao Wang. 2013. LensGesture: augmenting mobile interactions with back-of-device finger gestures. In *Proceedings of the 15th ACM on International conference on multimodal interaction*. ACM, 287–294.

[30] Ying Yin, Tom Ouyang, Kurt Partridge, and Shumin Zhai. 2013. Making touch-screen keyboards adaptive to keys, hand postures, and individuals: a hierarchical spatial backoff model approach. In *2013 ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*. Paris, France, 2775–2784.

[31] Quanzeng You, Hailin Jin, Zhaowen Wang, Chen Fang, and Jiebo Luo. 2016. Image Captioning with Semantic Attention. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, 4651–4659.

[32] Seunghak Yu, Nilesh Kulkarni, Haejun Lee, and Jihie Kim. 2018. On-Device Neural Language Model Based Word Prediction. In *The 27th International Conference on Computational Linguistics (COLING): System Demonstrations*. Santa Fe, NM, 128–131.

[33] Jingyuan Zhang, Xin Wang, Yue Feng, Mingming Sun, and Ping Li. 2018. FastInput: Improving Input Efficiency on Mobile Devices. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM)*. Torino, Italy, 2057–2065.